

UNITED STATES PATENT APPLICATION

FOR

DISTRIBUTED MEMORY MODULE CACHE

Applicant:

Howard S. David

Prepared by:

Calvin E. Wells
BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN
2200 Mission College Boulevard
Santa Clara, CA 95052

EXPRESS MAIL CERTIFICATE OF MAILING

"Express Mail" mailing label number EL867648947US

Date of Deposit December 31, 2001

I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Commissioner of Patents and Trademarks, Washington, D.C. 20231.

Michelle Begay

(Typed or printed name of person mailing paper or fee)

Michelle Begay
(Signature of person mailing paper or fee)

DISTRIBUTED MEMORY MODULE CACHE

Field Of The Invention

[0001] The present invention pertains to the field of computer systems. More particularly, this invention pertains to the field of memory system caches.

Background of the Invention

[0002] For a number of years, computer system memory subsystems have relied on a "stub bus" topology where memory modules are plugged into connectors on a memory bus. Each memory module adds a short electrical stub to the memory bus. Each memory module may contain several dynamic random access memory (DRAM) components and one or more buffer components electrically situated between the DRAM and memory bus connections. The stub bus topology is limited by signal integrity issues as to how fast data can be transferred over the memory bus.

[0003] In order to improve data throughput from the memory modules to a memory controller, some prior computer systems have used memory data caches. One type of prior cache involves a cache that is closely associated with the memory controller. The cache logic and tag memory along with the data cache are implemented at the memory controller end of the memory bus. One disadvantage of this type of cache is that it is generally only beneficial if the cache is several times larger than the caches associated with the computer system processor or processors. Another disadvantage of this type of cache is that valuable memory bus bandwidth is used to load lines of data from the memory modules to the cache. Because much of the loaded data will ultimately not be used, valuable memory bus bandwidth is wasted transferring unnecessary data.

[0004] Another type of prior cache system includes a data cache located on the DRAM devices themselves. The logic and tag memory may be located at the memory

controller end of the memory bus. These caches have the disadvantages of including a limited number of cache lines and also not storing the cached data any closer to the memory controller.

[0005] As more and more demands are placed on the memory subsystem, it will be desirable to implement a system memory cache that reduces read latencies and maximizes throughput while placing a minimum burden on memory bus bandwidth.

4005646 1.33.1.03

Brief Description of the Drawings

[0006] The invention will be understood more fully from the detailed description given below and from the accompanying drawings of embodiments of the invention which, however, should not be taken to limit the invention to the specific embodiments described, but are for explanation and understanding only.

[0007] Figure 1 is a block diagram of one embodiment of a computer system including a memory controller and a system memory.

[0008] Figure 2 is a block diagram of one embodiment of the memory controller including a cache tag and logic unit and the system memory including several memory modules.

[0009] Figure 3 is a block diagram of an example embodiment of the cache tag and logic unit.

[0010] Figure 4 is a block diagram of a tag unit included in the cache tag and logic unit.

[0011] Figure 5 is a block diagram of one embodiment of a data cache and logic unit located on a memory module.

[0012] Figure 6 is a flow diagram of one embodiment of a method for performing a read and preload command

Detailed Description

[0013] In general, one embodiment of a distributed memory module cache includes tag memory and associated logic implemented at the memory controller end of a memory channel. The memory controller is coupled to at least one memory module by way of a point-to-point interface. The data cache and associated logic are located in one or more buffer components on each of the memory modules.

[0014] Figure 1 is a block diagram of one embodiment of a computer system. The computer system includes a processor 110 and a memory controller hub 120. The memory controller hub 120 includes a memory controller 202. The memory controller 202 is coupled to a system memory 210.

[0015] The memory controller hub 120 is also coupled to an input/output hub 130. The input/output hub 130 provides communication with a peripheral component interconnect (PCI) bus 133 and a disk drive interface 131.

[0016] Figure 2 is a block diagram of one embodiment of the memory controller 202 including a cache tag and logic unit 300 and the system memory 210 including memory modules 220, 230, 240, and 250. The memory modules 220, 230, 240, and 250 include buffers 222, 232, 242, and 252, respectively. The buffers 222, 232, 242, and 252 include data cache and logic units 500, 238, 248, and 258, respectively. The memory module 220 includes DRAM components 223, 224, 225, and 226. The memory module 230 includes DRAM components 233, 234, 235, and 236. The memory module 240 includes DRAM components 243, 244, 245, and 246. The memory module 250 includes DRAM components 253, 254, 255, and 256.

[0017] In this example embodiment, the memory controller 202 is coupled to the memory modules 220, 230, 240, and 250 via a point-to-point interconnect 265. The interconnect 265 may include 8 differential pairs, 9 pairs for data, and 9 pairs for

address and command. The interconnect may transfer data at a rate several times the rate of the buffer to DRAM interconnect. Another embodiment may use 27 differential pairs, 18 pairs for data, and 9 pairs for address and command. The interconnect 265 may also include 18 differential pairs, 9 pairs for read data, and 9 pairs for write data, address and command. Still other embodiments are possible using a wide range of interconnect techniques and numbers of signals.

[0018] The memory controller 202 uses the cache tag and logic unit 300 to deliver commands and addresses to the memory modules 220, 230, 240, and 250 over the interconnect 265. The memory controller 202 also delivers and receives data to and from the memory modules 220, 230, 240, and 250.

[0019] Although this example embodiment shows buffers 222, 232, 242, and 252 as being located on the memory modules 220, 230, 240, and 250, respectively, other embodiments may implement these buffers on a motherboard instead of on the memory modules.

[0020] Whether on the motherboard or on the memory modules, the buffers may be implemented using more than one buffer component per memory module. Also, although this example embodiment shows four memory modules, other embodiments may include other numbers of memory modules. Further, other DRAM device configurations and numbers of DRAM components may be supported in other embodiments. Other embodiments are possible where the DRAM is placed on a motherboard rather than on a memory module. Still other embodiments are possible using types of memory devices other than DRAM.

[0021] The data cache and logic 500, 238, 248, and 258, along with the cache tag and logic 300, may comprise a 4 way set-associative cache. Other embodiments are possible using other numbers of ways. Some possible cache sizes include 288kB,

576kB, and 1152kB. Other embodiments are possible using different cache sizes.

The data cache and logic units 500, 238, 248, and 258 may include either embedded DRAM or static random access memory (SRAM).

[0022] The cache line size for these example embodiments may be 72 bytes.

The 72 bytes are transferred from the memory controller 202 to the memory modules 220, 230, 240, and 250 in bursts of eight transactions with 9 bytes being transferred per transaction. These example embodiments may also include an optional segmented cache. For a segmented cache, the cache line size is reduced to 36 bytes. The 36 bytes are transferred in bursts of four transactions with 9 bytes per transaction. The optionally segmented cache allows these example embodiments to support both memory devices that are optimized for bursts of eight and also memory devices that are optimized for bursts of four. By segmenting the cache when the cache line size is reduced to 36 bytes, the entire cache can be accessed without doubling the number of tags in the memory controller 202.

[0023] Figure 3 is a block diagram of an example embodiment of the cache tag and logic unit 300. The cache tag and logic unit 300 receives several groups of address information from the memory controller 202 which may receive this address information from a system agent such as the processor 110. One group of address lines includes A[35:27, 16-n:6] 301. The variable “n” may be “0” for a 288kB cache, or “1” for a 576kB cache, or “2” for a 1152kB cache. Another address group includes A[26:17-n] 303 and another address group includes the single line A[5] 305.

[0024] The address lines A[35:27, 16-n:6] 301 are delivered to a row/column/bank decode logic unit 310. From the address lines A[35:27, 16-n:6] 301, the row/column/bank decode logic unit 310 derives row addresses row[15:10+n] 311, column addresses col[11:0] 313, and DRAM bank information bank[2:0] 315.

[0025] The address lines A[35:27, 16-n:6] 301 are also delivered to a memory module decode logic unit 320. From the address line A[35:27, 16-n:6] 301, the memory module decode logic unit 320 decodes which memory module is being addressed and communicates this information via memory module[1:0] signals 317. The address lines A[35:27, 16-n:6] 301 are further coupled to a memory module 1 tag unit 400, a memory module 2 tag unit 340, a memory module 3 tag unit 350, and a memory module 4 tag unit 360.

[0026] The address lines A[26:17-n] 303 are used to create row addresses row[9+n:0] 309. The address lines A[26:17-n] 303 are also coupled to the memory module 1 tag unit 400, the memory module 2 tag unit 340, the memory module 3 tag unit 350, and the memory module 4 tag unit 360. The address line A[5] is also coupled to the memory module 1 tag unit 400, the memory module 2 tag unit 340, the memory module 3 tag unit 350, and the memory module 4 tag unit 360.

[0027] The memory module 1 tag unit 400, the memory module 2 tag unit 340, the memory module 3 tag unit 350, and the memory module 4 tag unit 360 each perform look-ups into their respective tag arrays to determine whether there is a cache hit and to determine whether data has been modified. The memory module tag units 400, 340, 350, and 360 also determine which cache way is currently implicated. The memory module 1 tag unit 400, and by extension the other tag units, will be discussed in more detail below.

[0028] The memory module 1 tag unit 400 communicates cache way information to a multiplexer (MUX) 370 via signals way[1:0] 319. Cache hit information is delivered to the MUX 370 over a hit 321 signal. Cache line modification information is communicated over a modified 323 signal. The memory module 2 tag unit 340 communicates cache way information to the MUX 370 via

signals way[1:0] 325. Cache hit information is delivered to the MUX 370 over a hit 327 signal. Cache line modification information is communicated over a modified 329 signal. The memory module 3 tag unit 350 communicates cache way information to the MUX 370 via signals way[1:0] 331. Cache hit information is delivered to the MUX 370 over a hit 333 signal. Cache line modification information is communicated over a modified 335 signal. The memory module 4 tag unit 360 communicates cache way information to the MUX 370 via signals way[1:0] 337. Cache hit information is delivered to the MUX 370 over a hit 339 signal. Cache line modification information is communicated over a modified 341 signal.

[0029] The MUX 370 receives the memory module[1:0] 317 signals and uses these signals to determine which of the way[1:0] signals 319, 325, 331, and 337 to pass on to a command sequencer and serializer unit 380 via way[1:0] 343 signals. Similarly, the state of the memory module[1:0] 317 signals determines which of the hit 321, 327, 333, and 339 signals are delivered to the command sequencer and serializer unit 380 over the hit 345 signal. The memory module[1:0] 317 signals also determine which of the modified signals 323, 329, 335, and 341 are communicated to the command sequencer and serializer unit 380 via the modified 347 signal.

[0030] The command sequencer and serializer unit 380 also receives the row[9+n:0] 309, row[15:10+n] 311, col[11:0] 313, bank[2:0] 315, and memory module[1:0] 317 signals. The command sequencer and serializer unit 380 uses the inputs described above in addition to other communications from the memory controller 202 to generate a series of command signals command[3:0] 351 and address signals addr[4:0] 353. The command[3:0] 351 and addr[4:0] 353 signals are delivered to the memory modules 220, 230, 240, and 250 over the interconnect 265. A

discussion of some of the possible commands for these example embodiments appears below.

[0031] In the cache tag and logic unit 300, the actions of the tag look-up and memory module decode occur in parallel. That is, while the memory module decode logic 320 is determining the proper memory module for the current transaction, the memory module tag units 400, 340, 350, and 360 are performing their look-ups. Different sub-sets of the address information are used to perform the tag look-ups and the memory module decode. For this example embodiment, the address lines A[26:17-n] are used as an index into the tag arrays (see Figure 4, below). The address lines A[35:27, 16-n:6] are used to perform the memory module decode.

[0032] Figure 4 is a block diagram of the memory module 1 tag unit 400. The address lines A[26:17-n] 303 are received at a tag way 1, tag way 2, tag way 3, and tag way 4 units. These tag way units contain arrays of tag addresses. Because the cache for this example embodiment is a 4-way set associative cache, there are tag arrays for each of the 4 ways. The address lines A[26:17-n] 303 provide an index into the tag way units 1, 2, 3, and 4. The cache lines associated with the index for the tag way units 1, 2, 3, and 4 are placed on tag data[21:0] lines 401, 407, 413, and 419 respectively. The tag data[21:0] 401, 407, 413, and 419 are delivered to comparators 410, 420, 430, and 440, respectively.

[0033] The comparators 410, 420, 430, and 440 compare the received tag data with the address lines A[35:27, 16-n:6] 301 to determine whether there is a tag match and therefore a cache hit. The tag match information is communicated with a logic unit 450. The logic unit 450 also receives valid[1:0] signals 403, 409, 415, and 421 from the tag way units 1, 2, 3, and 4 respectively. When the cache is set up as a segmented cache, the valid[1:0] signals 403, 409, 415, and 421 indicate which cache

lines associated with a tag address are valid. The state of the address line A[5] 305 indicates to the logic unit 450 which of the two valid bits are appropriate. The logic unit 450 further receives modified signals 405, 411, 417, and 423 from the tag way units 1, 2, 3, and 4, respectively. With this information, the logic unit 450 is able to generate the way[1:0] signals 319 which indicate which way is implicated by the current transaction. The logic unit 450 also generates the hit signal 321 which indicates if there has been a tag match. The logic unit 450 further generates the modified signal 323.

[0034] Figure 5 is a block diagram of one embodiment of the data cache and logic unit 500. The data cache and logic unit 500 receives the command[3:0] signals 351 and the addr[4:0] lines 353 at a command deserializer and decoder unit 510. The command deserializer and decoder unit 510 processes the received command and address information and provides control for an eviction buffer 540, a cache data unit 530, row address latches 520, and MUXs 550, 560, and 570. The MUXs 550, 560, and 570 are controlled via MUX control signals 517. The row address latches 520 receive row addresses row[9+n:0] 509 from the command deserializer and decoder unit 510. The command deserializer and decoder unit 510 also delivers latch enable signals LE[3:0] 511 and bank address lines bank[2:0] 513 to the row address latches 520. The row address latches 520 in turn deliver a selected row address to the cache data unit 530. The eviction buffer 540 receives output data 519 from the cache data unit 530. When cache lines are evicted out of the cache data 530, the lines are stored temporarily in the eviction buffer 540. The eviction buffer 540 is controlled via eviction control signals 503 from the command deserializer and decoder unit 510.

[0035] The example embodiments described herein implement a write-back policy. That is, write transactions from the memory controller 202 to the system

memory 210 may be stored in the memory module data caches until evicted from the cache. The evicted cache lines are stored in an eviction buffer until the memory controller 202 determines that the DRAM for that memory module is not busy and signals a writeback command. The cache line is then delivered from the eviction buffer to the DRAM.

[0036] The write-back cache allows write data to be held until a time that the DRAM interconnect is not busy. The data stays in the cache until the memory controller signals an eviction, which moves that cache line to the eviction buffer. If the row for that cache line is not open, an activate command is issued (the activate command is discussed below). Lastly, the writeback command supplies way and bank information in order to write the data in to the DRAM. In two and four processor systems, it is common for some cache lines to move back and forth between processors, with a write to memory for each transfer. If a line is being moved back and forth frequently, the write-back cache will reduce the amount of writes to the DRAM.

[0037] Data that is transferred over the interconnect 265 is assembled or disassembled by the buffer 222. The buffer 222 delivers data[71:0] 501 to the MUX 560 and to the MUX 550. The MUX 560 also receives the output from the eviction buffer 540. The MUX 560 passes either the data[71:0] signals 501 or the output of the eviction buffer 540 to the DRAM devices depending on the state of the MUX control signals 517.

[0038] The MUX 570 receives data from the DRAM devices and also receives data out[71:0] signals 519 from the cache data 530. The MUX 570 passes either the data from the DRAM devices or the data out[71:0] 519 to the buffer 222 depending on the state of the MUX control signals 517.

[0039] As previously mentioned, the buffer 222 delivers data[71:0] 501 to the MUX 560 and to the MUX 550. The MUX 550 also receives data from the DRAM devices. The MUX 550 passes either the data[71:0] signals 501 or the data from the DRAM devices to the cache data 530 via the data in[71:0] signals 521.

[0040] The memory controller 202 may cause a number of commands to be executed by the data cache and logic unit 500. These commands are communicated by the command sequencer and serializer unit 380 to the data cache and logic unit 500 using the command[3:0] signals 351 and the addr[4:0] signals 353. Some of the possible commands include activate, read, write, cache fetch, cache load, read and load, write and load, read and preload, read and load double, writeback, and refresh.

[0041] The activate command involves storing a row address in one of the banks of the DRAM devices on a memory module and activating the sense amplifiers for that row of that bank in those DRAM, so that the data in that row may be accessed. A new row address is also loaded into the row address latches 520. Table 1 describes how the command sequencer and serializer unit 380 transfers the activate command to the memory modules.

Signal	Activate Command			
	Transfer			
	1	2	3	4
Command 0	Dest 0	RAS=1		Hit=0
Command 1	Dest 1	CAS=0		
Command 2		WE=0		
Command 3				
Addr 0	RA0	RA4	RA8	RA12
Addr 1	RA1	RA5	RA9	RA13
Addr 2	RA2	RA6	RA10	RA14
Addr 3	RA3	RA7	RA11	RA15
Addr 4	BA0	BA1	BA2	

Table 1 - Activate Command

[0042] During the first transfer period, the command[1:0] lines carry a destination designation (Dest) that identifies to which memory module the command is intended. During the second transfer period the command[2:0] lines carry state information for a row address strobe (RAS), a column address strobe (CAS), and a write enable signal (WE). During the fourth transfer period, the command 0 line gives an indication that there is no cache hit.

[0043] The addr[3:0] lines provide row address (RA) information during the four transfer periods. The addr 4 line provides bank address information (identifying a DRAM bank) during the first three transfer periods. The bank address information is used to store a new row address into the row address latches 520.

[0044] The cache fetch command involves reading a line of data out of the cache of the identified memory module. Table 2 describes how the command sequencer and serializer unit 380 transfers the cache fetch command to the memory modules.

	Cache Fetch Command			
	Transfer			
Signal	1	2	3	4
Command 0	Dest 0	RAS=1		Hit=1
Command 1	Dest 1	CAS=0		Way0
Command 2		WE=0		Way1
Command 3				
Addr 0	RA0	RA4	RA8	RA12
Addr 1	RA1	RA5	RA9	RA13
Addr 2	RA2	RA6	RA10	RA14
Addr 3	RA3	RA7	RA11	RA15
Addr 4	BA0	BA1	BA2	

Table 2 - Cache Fetch Command

[0045] The cache fetch command information is much the same as that for the activate command. However, in the fourth transfer period, the command 0 line carries an indication of a cache hit. The command[2:1] lines carry information indicating which cache way is implicated by the cache hit during the fourth transfer period. By providing the cache hit and way information in the last transfer period, the cache tag and logic unit 300 has the maximum amount of time to perform a tag look-up and to determine which command to transfer. Notice that the only differences between the activate command and the cache fetch command are the hit and way information in the last transfer period.

[0046] The read command causes data to be read out of DRAM without being loaded into the cache. Table 3 describes how the command sequencer and serializer unit 380 transfers the read command to the memory modules.

	Read Command			
	Transfer			
Signal	1	2	3	4
Command 0	Dest 0	RAS=0		Hit=0
Command 1	Dest 1	CAS=1		
Command 2		WE=0		
Command 3				
Addr 0	CA0	CA4	CA8	CA12
Addr 1	CA1	CA5	CA9	CA13
Addr 2	CA2	CA6	CA10	
Addr 3	CA3	CA7	CA11	
Addr 4	BA0	BA1	BA2	

Table 3 - Read Command

[0047] During the first transfer period, the command[1:0] lines carry a destination designation (Dest) that identifies to which memory module the command is intended. During the second transfer period the command[2:0] lines carry state information for a row address strobe (RAS), a column address strobe (CAS), and a

write enable signal (WE). During the fourth transfer period, the command 0 line gives an indication that there is no cache hit.

[0048] The addr[3:2] lines provide column address (CA) information during the first three transfer periods. The addr[1:0] lines provide column address information during each of the four transfer periods. The addr 4 line provides bank address information during the first three transfer periods. The bank address information is used to select a previously stored row address from the row address latches 520.

[0049] The read and preload command involves reading a line of data out of the DRAM devices of the identified memory module and then preloading an additional line of data into the data cache. Table 4 describes how the command sequencer and serializer unit 380 transfers the read and preload command to the memory modules.

	Read and Preload Command			
	Transfer			
Signal	1	2	3	4
Command 0	Dest 0	RAS=0		Hit=1
Command 1	Dest 1	CAS=1		Way 0
Command 2		WE=0		Way 1
Command 3				Evict
Addr 0	CA0	CA4	CA8	CA12
Addr 1	CA1	CA5	CA9	CA13
Addr 2	CA2	CA6	CA10	
Addr 3	CA3	CA7	CA11	
Addr 4	BA0	BA1	BA2	

Table 4 - Read and Preload Command

[0050] The read and preload command information is much the same as that for the read command. However, in the fourth transfer period, the command 0 line

carries an indication of a cache hit. The command[2:1] lines carry information indicating which cache way is implicated by the cache hit during the fourth transfer period. By providing the cache hit and way information in the last transfer period, the cache tag and logic unit 300 has the maximum amount of time to perform a tag look-up and to determine which command to transfer. The command 3 line includes control information for the eviction buffer 540. Notice that the only differences between the read command and the read and preload command are the hit, way, and eviction information in the last transfer period.

[0051] Figure 6 is a flow diagram of one embodiment of a method for fetching data from a memory module that includes a data cache. At block 610, a read request is received at a memory controller. At block 620 a tag look-up is performed to determine whether there is a cache hit. Block 630 indicates that if there is a cache hit, a line of cache data is fetched from the data cache located on the memory module at block 640. Block 630 also indicates that if there is a cache miss, a line of data is loaded from a memory device on the memory module to the data cache at block 650. At block 660, the line of data is delivered to the memory controller.

[0052] In the foregoing specification the invention has been described with reference to specific exemplary embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention as set forth in the appended claims. The specification and drawings are, accordingly, to be regarded in an illustrative rather than in a restrictive sense.

[0053] Reference in the specification to "an embodiment," "one embodiment," "some embodiments," or "other embodiments" means that a particular feature, structure, or characteristic described in connection with the embodiments is included

